View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Kyutacar : Kyushu Institute of Technology Academic Repository

九州工業大学学術機関リポジトリ
Kyutacar
Kyushu Institute of Technology Academic Repository

On NIS-Apriori Based Data Mining in SQL

著者	Sakai Hiroshi, Liu Chenxi, Zhu Xiaoxin, Nakata Michinori
journal or publication title	Lecture Notes in Computer Science
volume	9920
page range	514-524
year	2016-09-29
URL	http://hdl.handle.net/10228/00006391

doi: info:doi/10.1007/978-3-319-47160-0_47

On NIS-Apriori based Data Mining in SQL

Hiroshi Sakai[1], Chenxi Liu[1], Xiaoxin Zhu[2], and Michinori Nakata[3]

[1] Graduate School of Engineering, Kyushu Institute of Technology
Tobata, Kitakyushu 804-8550, Japan
sakai@mns.kyutech.ac.jp, p350932s@mail.kyutech.jp
[2] Graduate School of of Engineering, Harbin Institute of Technology
Westdazhi Str. Nangang Dist. Harbin City, China
XiaoxinZhu@hit.edu.cn
[3] Faculty of Management and Information Science,
Josai International University
Gumyo, Togane, Chiba 283-0002, Japan
nakatam@ieee.org

Abstract. We have proposed a framework of *Rough Non-deterministic Information Analysis* (RNIA) for tables with non-deterministic information, and applied RNIA to analyzing tables with uncertainty. We have also developed the RNIA software tool in Prolog and *getRNIA* in Python, in addition to these two tools we newly consider the RNIA software tool in SQL for handling large size data sets. This paper reports the current state of the prototype named *NIS-Apriori in SQL*, which will afford us more convenient environment for data analysis.

Keywords: Association rules, NIS-Apriori algorithm, SQL, Prototype, Uncertainty.

1 Introduction

We have been coping with rough sets [7], non-deterministic information [6, 7], the *Apriori* algorithm [1, 12], the software tool in Prolog [9], and *getRNIA* in Python [11, 15]. Recently, we are considering a software tool in SQL in order to handle large size data sets.

In rough sets, we usually employ *Deterministic Information Systems* (*DISs*) with deterministic attribute values. We can see every *DIS* is a standard table. For handling information incompleteness in tables [2, 4, 6, 7, 9], we employ *Non-deterministic Information Systems* (*NISs*) with non-deterministic values and missing values. By changing *DIS* to *NIS*, several new issues occurred, for example the *possible equivalence classes*, the *minimum* and the *maximum degrees of data dependency*, the *certain* and the *possible* rules, and so on [9]. At the same time, one computational problem occurred, namely the computational complexity may increase exponentially due to the case analysis on *NIS*. However, in rule generation, we proved some properties and escaped from the exponential order problem [10, 11]. Due to this result, the rule generator in Prolog [10] and *getRNIA* in Python [15] were implemented.

In this paper, we focus on the rule generator in SQL, because SQL has the high versatility. Furthermore, several algorithms including *Apriori* were investigated in [12]. For handling large size data sets, we think SQL will be more suitable than the previous languages, Prolog and Python. Recently, the 'sparse' property of the data sets is considered [2, 14]. The density of the important part in the data sets may not be unique, and we may ignore the meaningless part. In the sparse matrix, we may employ the special format for reducing the data size. The use of this sparse property will be another approach to large size data sets.

As for this work, we need to specify that this is not the first trial, and the first trial was done in [13]. We follow the result in [13], and consider a rule generator which we name *NIS-Apriori in SQL*.

This paper is organized as follows: Section 2 surveys RNIA and rule generation. Section 3 investigates *NIS-Apriori* in SQL and its prototype system. Section 4 concludes this paper.

2 RNIA and Rule Generation

At first, we clarify the rules in *DIS*. For a fixed decision attribute *Dec* and a set *CON* of attributes, we see an implication $\tau : \wedge_{A \in CON}[A, val_A] \Rightarrow [Dec, val]$ is (a candidate of) a *rule*, if τ satisfies the next two constraints.

$$\begin{array}{l} \text{For two threshold values } 0 < \alpha, \beta \leq 1.0, \\ support(\tau)(= N(\tau)/|OB|) \geq \alpha, \\ accuracy(\tau)(= N(\tau)/N(\wedge_{A \in CON}[A, val_A])) \geq \beta, \\ \text{Here, } N(*) \text{ means the number of objects satisfying} \\ \text{the formula } *, \ OB \text{ means a set of all objects.} \end{array} \quad (1)$$

Then, we briefly survey rule generation in RNIA. Figure 1 shows *NIS* Ψ_1, where we see [high,veryhigh] and nil. Here, [high,veryhigh] is non-deterministic information, namely either high or veryhigh is the actual value, and nil is missing value. Each nil may take every possible value in the attribute.

In Ψ_1, we replace each non-deterministic information and nil with a possible value, and we obtain a table with deterministic information. We named it a *derived DIS* from *NIS*. Let $DD(\Psi)$ be a set of all derived *DISs* from Ψ. We see an actual *DIS* ϕ^{actual} exists in $DD(\Psi)$. For Ψ_1, $DD(\Psi_1)$ consists of 4608 $(=3^2 \times 2^9)$ derived *DISs*. Based on $DD(\Psi)$, we proposed the certain and the possible rules below:

Definition 1. *[10] For NIS Ψ and the decision attribute Dec, we fix the threshold values α and β $(0 < \alpha, \beta \leq 1.0)$.*
(1) We say τ is a certain rule, if τ satisfies $support(\tau) \geq \alpha$ and $accuracy(\tau) \geq \beta$ in each $\phi \in DD(\Psi)$,
(2) We say τ is a possible rule, if τ satisfies $support(\tau) \geq \alpha$ and $accuracy(\tau) \geq \beta$ in at least one $\phi \in DD(\Psi)$.

Definition 1 seems natural, but we have the computational complexity problem, because the number of elements in $DD(\Psi)$ increases exponentially. In Ψ_1,

object	temp(erature)	head(ache)	nau(sea)	flu
x1	high	nil	no	yes
x2	[high,veryhigh]	yes	yes	yes
x3	nil	no	no	nil
x4	high	yes	nil	nil
x5	high	nil	yes	no
x6	normal	yes	nil	nil
x7	normal	no	yes	no
x8	nil	yes	nil	yes

Fig. 1. An exemplary *NIS* Ψ_1.

object	attrib	value	det
x1	temp	high	1
x1	head	yes	2
x1	head	no	2
x1	nau	no	1
x1	flu	yes	1
x2	temp	high	2
x2	temp	veryhigh	2
x2	head	yes	1

Fig. 2. A part of Ψ_1 in NRDF format.

the number is 4608, and the number is more than 10^{100} in Mammographic data set in UCI machine learning repository [3]. For this computational problem, we defined two sets for a descriptor $[A, val]$ below:

$inf([A, val])$=$\{x : object \mid$ the value of x for A is a singleton set $\{val\}\}$,
$sup([A, val])$=$\{x : object \mid$ the value of x for A is a set including $val\}$,
$inf(\wedge_{A\in CON}[A, val_A])$=$\cap_{A\in CON} inf([A, val_A])$,
$sup(\wedge_{A\in CON}[A, val_A])$=$\cap_{A\in CON} sup([A, val_A])$.

For example, $inf([head, yes])$=$\{x2, x4, x6, x8\}$ and $sup([head, yes])$=$inf([head, yes]) \cup \{x1, x5\}$ hold in Ψ_1. The actual equivalence class is between two sets. For *NIS* Ψ, an implication τ, and $minsupp(\tau)$ and $minacc(\tau)$ defined by $min_{\phi\in DD(\Psi)}\{support(\tau)$ by $\phi\}$ and $min_{\phi\in DD(\Psi)}\{accuracy(\tau)$ by $\phi\}$, we have the following which do not depend upon the number of $DD(\Psi)$.

$$\begin{array}{l} \tau : \wedge_{A\in CON}[A, val_A] \Rightarrow [Dec, val], \\ minsupp(\tau) = |inf(\wedge_{A\in CON}[A, val_A]) \cap inf([Dec, val])|/|OB|, \\ minacc(\tau) = \frac{|inf(\wedge_{A\in CON}[A, val_A]) \cap inf([Dec, val])|}{|inf(\wedge_{A\in CON}[A, val_A])| + |OUTACC|}, \\ OUTACC = \{sup(\wedge_{A\in CON}[A, val_A]) \setminus inf(\wedge_{A\in CON}[A, val_A])\} \\ \quad \setminus inf([Dec, val]). \end{array} \tag{2}$$

The $OUTACC$ means a set of objects, from which we can obtain an implication $\tau' : \wedge_{A\in CON}[A, val_A] \Rightarrow [Dec, val']$ $(val \neq val')$. Similarly, we can calculate $maxsupp(\tau)$ and $maxacc(\tau)$. We can also prove that there exists $\phi_{min} \in DD(\Psi)$ which makes both $support(\tau)$ and $accuracy(\tau)$ the minimum. There exists $\phi_{max} \in DD(\Psi)$ which makes both $support(\tau)$ and $accuracy(\tau)$ the maximum. Based on these results, we have the chart in Figure 3 and Theorem 1.

Theorem 1. *For an implication τ, we have the following.*
(1) τ is a certain rule, if and only if $minsupp(\tau) \geq \alpha$ and $minacc(\tau) \geq \beta$.
(2) τ is a possible rule, if and only if $maxsupp(\tau) \geq \alpha$ and $maxacc(\tau) \geq \beta$.

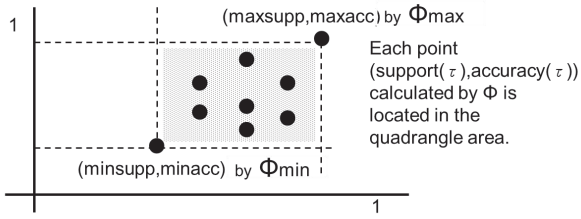

Fig. 3. The distribution of each point $(support(\tau), accuracy(\tau))$ by $\phi \in DD(\Psi)$.

(3) Even though the certain rules and the possible rules depend upon $DD(\Psi)$, the conditions to check them do not depend upon $DD(\Psi)$.

Based on Theorem 1, we can escape from the exponential order problem. Without Theorem 1, it will be hard to handle Mammographic data set, which has more than 10^{100} derived *DISs*.

3 NIS-Apriori in SQL

3.1 NIS-Apriori Algorithm

The *Apriori* algorithm was proposed by Agrawal, and this is the representative algorithm in data mining [1, 12]. This algorithm handles transaction data, and each transaction is given as a set of items. We identify each descriptor in table data with an item, then we can consider the *Apriori* algorithm in tables [10, 11]. In certain rule generation, we compare the minimum point in Figure 3 with the threshold values α and β. On the other hand, we compare the maximum point in Figure 3 with the threshold values α and β. Since the management of the implications is almost the same as in case of the *Apriori* algorithm and the calculation does not depend upon $|DD(\Psi)|$, we figure out that the computational complexity of the *NIS-Apriori* algorithm is about twice the complexity of the *Apriori* algorithm.

3.2 The NRDF Format

In data sets, we usually have the csv format. This is very familiar, however the name of the attribute and the number of all attributes may be different in each data set. For handling various types of data sets, it is useful to employ another unified format. Otherwise, the program is depending upon the number of the attributes and the name of the attribute.

Based on [13], we employ the NRDF format, which is the extended RDF (resource description framework) format, for any data set. This RDF format

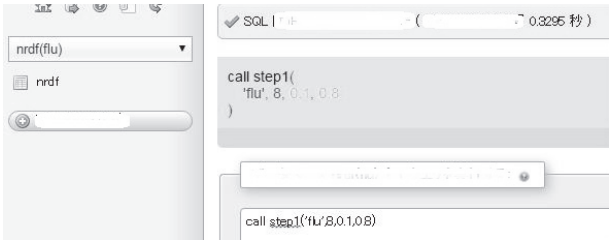

Fig. 4. SQL query execution, where Japanese characters were erased.

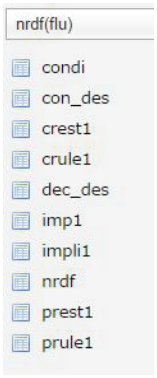

Fig. 5. All created tables.

may be called as the EAV (entity-attribute-value) format [5, 14]. In [5], the KDD-related tasks of attribute selection and decision tree induction were implemented based on the EAV format.

The NRDF format employs 4 attributes, *object*, *attrib*, *value*, and *det*. Figure 2 shows a part of the NRDF format of Ψ_1. In order to specify non-deterministic information, we added the 4th column *det*. The value of *det* means the number of possible values. If *det*=1, this means that the value is deterministic. Otherwise, we know the value is non-deterministic and the number of values by *det*.

3.3 Step 1: Rule Generation in the Form of $P_1 \Rightarrow Dec$

In Step 1, the procedure *step*1 generates the certain and the possible rules in the form of $P_1 \Rightarrow Dec$. This procedure consists of the following:

```
1. create table condi (the condition of the rules),
2. create table con_des (the descriptors for the condition),
3. create table dec_des (the descriptors for the decision),
4. create table impli1 (the implications with inf, sup, inacc, outacc),
5. create table crule1 (the certain rules),
6. create table prule1 (the possible rules),
7. create table crest1 (the candidates of Step 2),
8. create table prest1 (the candidates of Step 2).
```

At first, a file *nrdf* in the NRDF format in Figure 2 is stored in the system (Figure 4). In Figure 4, we execute '*call step*1$('flu', 8, 0.1, 0.8)$', which means the decision attribute is $'flu'$, the number of the objects is 8, the *support* value is 0.1, and the *accuracy* value is 0.8. It took about 0.33 (sec) for executing the procedure *step*1 in windows PC, and all tables in Figure 5 were generated.

In Figure 5, two tables *con_des* and *dec_des* store the set of descriptors on the condition part and the set of descriptors on the decision part, respectively.

att1	val1	deci	deci_value	inf	sup	inacc	outacc
head	no	flu	no	1.000	3.000	1.000	1.000
head	no	flu	yes	0.000	2.000	1.000	1.000
head	yes	flu	no	0.000	3.000	1.000	1.000
head	yes	flu	yes	2.000	5.000	1.000	1.000
nau	no	flu	no	0.000	3.000	2.000	3.000
nau	no	flu	yes	1.000	5.000	3.000	2.000
nau	yes	flu	no	2.000	4.000	2.000	3.000
nau	yes	flu	yes	1.000	4.000	3.000	2.000

Fig. 6. A part of *impli*1.

att1	val1	deci	deci_value	minsupp	minacc
head	no	flu	no	0.125	0.333
head	yes	flu	yes	0.250	0.400
nau	no	flu	yes	0.125	0.250
nau	yes	flu	no	0.250	0.333
nau	yes	flu	yes	0.125	0.200
temp	high	flu	no	0.125	0.167
temp	high	flu	yes	0.125	0.250
temp	normal	flu	no	0.125	0.250

Fig. 7. Total contents in *crest*1.

The procedure *step*1 generates the Cartesian Products by using *con_des* and *dec_des*, and adds *inf*, *sup*, *inacc* and *outacc* to the table *impli*1 (Figure 6).

Based on *impli*1, the procedure *step*1 calculates *minsupp* and *minacc* for each tuple and compares them with the threshold values α and β. If $minsupp \geq \alpha$ and $minacc \geq \beta$, the procedure *step*1 adds this tuple to the table *crule*1. If $minsupp \geq \alpha$ and $minacc < \beta$, the procedure adds this tuple to the table *crest*1 (Figure 7). On the other hand, the procedure calculates *maxsupp* and *maxacc* for each tuple and compares them with the threshold values α and β. If $maxsupp \geq \alpha$ and $maxacc \geq \beta$, the procedure adds this tuple to the table *prule*1 (Figure 8). If $maxsupp \geq \alpha$ and $maxacc < \beta$, the procedure adds this tuple to the table *prest*1. The following is the SQL procedure for generating the table *prule*1.

The procedure for *prule*1 in Step 1:

```
create table prule1 (att1 varchar,val1 varchar,deci varchar,
   deci_value varchar,maxsupp decimal,maxacc decimal)
select impli1.att1,impli1.val1,impli1.deci,impli1.deci_value,
   impli1.sup/ob as maxsupp,impli1.sup/(con_des.inf+inacc) as maxacc
from impli1,con_des
where impli1.att1=con_des.attrib and impli1.val1=con_des.value
having maxsupp >=alpha and maxacc >=beta;
```

In Step 1, the most complicated part is to generate the table *impli*1. After obtaining the Cartesian Products *imp*1, *step*1 sequentially adds *inf*, *sup*, *inacc*, and *outacc* to *impli*1. If $inf([A, val_A] \wedge [Dec, val])$ is an empty set, this tuple is not stored in the temporary table data set. Even though it is necessary to add inf=0 to the table *impli*1, the value NULL is added to *impli*1 in this case. Therefore, *step*1 replaces NULL with 0 after adding *inf* information to *impli*1. The same occurs for *sup*, *inacc*, and *outacc*. In the current implementation, we faithfully simulated the *NIS-Apriori* algorithm, and there are ineffective procedures including the above case. It is necessary to reduce such ineffective part.

att1	val1	deci	deci_value	maxsupp	maxacc
head	no	flu	no	0.375	1.000
head	yes	flu	yes	0.625	1.000
nau	no	flu	yes	0.625	1.000
nau	yes	flu	no	0.500	0.800
temp	high	flu	yes	0.625	0.833
temp	normal	flu	no	0.375	1.000
temp	veryhigh	flu	no	0.125	1.000
temp	veryhigh	flu	yes	0.375	1.000

Fig. 8. Total contents in *prule*1.

a1	v1	a2	v2	deci	deci_value	minsupp	minacc
head	no	nau	yes	flu	no	0.125	1.000

Fig. 9. Total contents in *crule*2.

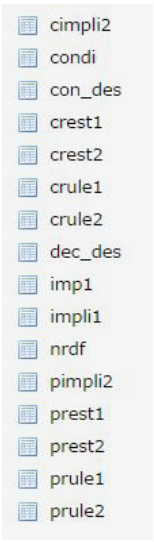

Fig. 10. All created tables.

3.4 Step 2: Rule Generation in the Form of $P_1 \wedge P_2 \Rightarrow Dec$

In Step 2, the procedure *step*2 generates the certain and the possible rules in the form of $P_1 \wedge P_2 \Rightarrow Dec$. Since $support(P_1 \wedge P_2 \Rightarrow Dec) \leq support(P_1 \Rightarrow Dec)$ holds, it is enough to consider the implications $P_1 \wedge P_2 \Rightarrow Dec$ satisfying $(P_1 \Rightarrow Dec), (P_2 \Rightarrow Dec) \in crest1$ in certain rule generation.

We execute '$call\ step2('flu', 8, 0.1, 0.8)$' again, and it took about 0.39 (sec) for executing the procedure *step*2. Then, all tables in Figure 10 were generated. In Figure 10, two tables *cimpli*2 and *pimpli*2 store the tuples with *inf*, *sup*, *inacc* and *outacc*, respectively. Tables *crule*2 and *prule*2 store the certain rules and the possible rules in the form of $P_1 \wedge P_2 \Rightarrow Dec$. Similarly to the tables *crest*1 and *prest*1, *crest*2 and *prest*2 are generated for Step 3. In Step 2, we obtained a certain rule in Figure 9 and 12 possible rules in *prule*2.

The rule generation in Step 3 is the same as in case of Step2. Like Step 2, we execute '$call\ step3('flu', 8, 0.1, 0.8)$', then the procedure *step*3 generates rules.

3.5 An Implementation of NIS-Apriori in SQL

This prototype system is implemented on desktop PC and note PC by using the phpMyAdmin tool [8]. Currently, we made three procedures *step*1, *step*2, and *step*3 by using SQL command procedures. The data size of this file including all procedures is about 40KB in the text format. Since SQL command procedure is familiar, we will be able to use this prototype in the most of PC with SQL. Actually, we employed both desktop PC and note PC simultaneously for this

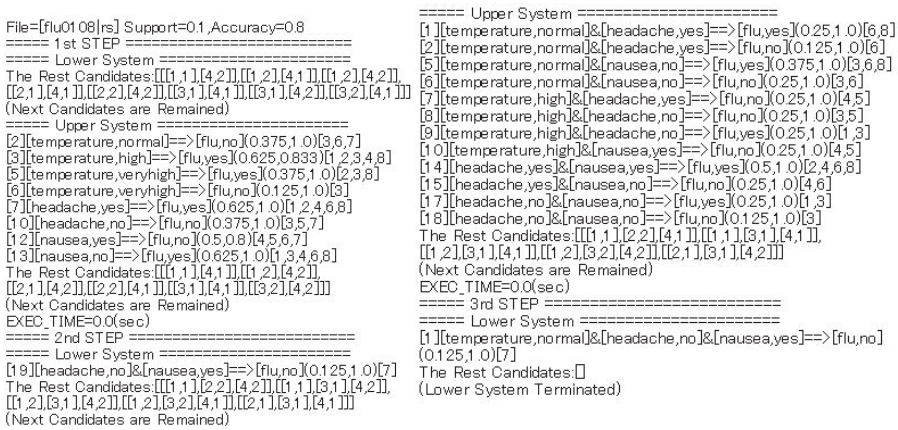

Fig. 11. The execution log by RNIA in Prolog.

implementation. We can also handle any *DIS* as a special case of *NIS*. In the NRDF format, we specify det=1 in each tuple, then we have the same rules in certain rule generation and possible rule generation.

3.6 The Difference between Two Software Tools RNIA in Prolog and NIS-Apriori in SQL

Figure 11 is the execution log for *NIS* Ψ_1 by RNIA in Prolog. Except the redundant case of the rules, we examined the result by RNIA in Prolog is equal to the result by *NIS-Apriori* in SQL. This will be an assurance that two software tools were implemented correctly.

Now, we have to remark the difference between the data structures of two software tools. RNIA in Prolog employs two blocks inf and sup, and internally manages them for each calculation. On the other hand, *NIS-Apriori* in SQL does not employ them directly, and employs the total search of the data set. These two points are the big difference between two software tools. We explain these two points below.

RNIA in Prolog generates $inf([A, val_A])$ and $sup([A, val_A])$ information for each descriptor $[A, val_A]$, and $inf(\tau)$ and $sup(\tau)$ are generated for each τ. For example, the set $inf([A, val_A] \wedge [Dec, val])$ is defined by $inf([A, val_A]) \cap inf([Dec, val])$, and it is stored as a temporary set. RNIA in Prolog makes use of $inf(\tau)$ and $sup(\tau)$, and generates rules. However, the use of $inf(\tau)$ and $sup(\tau)$ for each τ may be a heavy load. Figure 12 is the beginning of the log data for Mammographic data set [3]. In Figure 12, we see that 427 objects support this rule, and every number of the object, i.e., 3, 5, $\cdots$, 960, is stored in the list. Even though Prolog has a list processing functionality, the manipulation of such large size lists will be a heavy load.

On the other hand, *NIS-Apriori* in SQL does not store every number of the object, but stores the amount of objects (Figure 13). For obtaining $inf(\tau)$ and

```
File=[mammo0204|rs] Support=0.2,Accuracy=0.4
===== 1st STEP ========================
===== Lower System ====================
[1] [assessment,4]==>[severity,0](0.445, 0.763)
[3,5,6,12,13,14,15,19,22,30,33,34,35,36,39,40,41,42,43,47,50,52,53,55,56,58,62,
65,66,68,69,70,75,77,80,83,85,88,92,96,97,99,101,102,103,104,105,108,110,113,
114,115,116,117,121,122,123,125,126,127,128,133,138,141,142,143,144,149,152,
153,155,157,158,161,162,163,164,167,170,171,172,173,174,176,177,180,182,183,
    :   :   :   :   :   :
909,911,913,915,916,918,919,920,922,925,927,929,930,932,933,934,936,937,938,
939,941,943,945,946,947,949,953,954,956,958,960]
```

Fig. 12. The list of the objects supporting an implication.

a1	v1	a2	v2	a3	v3	deci	deci_value	infC	sup	inacc
age	50	assess	4	density	3	severity	0	111.000	100.000	10.000
age	50	density	3	margin	4	severity	0	45.000	0.000	7.000
age	60	assess	4	density	3	severity	0	125.000	107.000	15.000
age	60	density	3	margin	4	severity	0	75.000	0.000	10.000
assess	4	density	3	margin	4	severity	1	97.000	0.000	12.000

Fig. 13. Total contents in *cimpli*2 for Mammographic data set.

$sup(\tau)$, *NIS-Apriori* in SQL executes the total search in the NRDF data set. As we have described, the most complicated part is to add inf, sup, $inacc$, and $outacc$ information to the Cartesian Products. For this part, we need to employ the total search of the NRDF data set instead of manipulating $inf(\tau)$ and $sup(\tau)$, but we can escape from the manipulation on the large size lists. In the application of *NIS-Apriori* in SQL to Mammographic data set, we obtained the same result by RNIA in Prolog. However, the execution time by the implemented *NIS-Apriori* in SQL was not good. It took about 1 (min) for Step 1. It is necessary to revise the current procedure, especially the generation of *impli*1, *cimpli*2, and *plimpli*2.

4 Concluding Remarks

This paper briefly described the background of RNIA for handling information incompleteness in table data, and we newly focused on SQL system for handling large size data sets. As for this prototype, we have the following consideration.
(1) Since SQL has the high versatility, *NIS-Apriori* in SQL will offer the useful environment for analyzing tables with non-deterministic values.
(2) Both RNIA in Prolog and *getRNIA* in Python internally store a list for each implication. For large size data sets, the manipulation of these lists will be a heavy load. On the other hand, *NIS-Apriori* in SQL does not employ such lists, but it employs the total search of the data sets. In two strategies, i.e., the list manipulation strategy and the total search strategy, we figure out that the list

manipulation strategy will be suitable to rule generation for small size data sets, and the total search strategy will be suitable to rule generation for large size data sets.
(3) In the prototype, we faithfully simulated the *NIS-Apriori* algorithm, so the procedures in SQL might generate the meaningless tables. It is necessary to revise this point.

Acknowledgment: The authors would be grateful to the anonymous referees for their useful comments. This work is supported by JSPS (Japan Society for the Promotion of Science) KAKENHI Grant Number 26330277.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. *Proc. VLDB'94*, Morgan Kaufmann, 487–499 (1994)
2. Clark, P., Grzymala-Busse, J.: Mining incomplete data with many attribute-concept values and "do not care" conditions. *Proc. IEEE Big Data 2015*, 1597–1602 (2015)
3. Frank, A., Asuncion, A.: UCI machine learning repository. Irvine, CA: University of California, School of Information and Computer Science (2010) `http://mlearn.ics.uci.edu/MLRepository.html`
4. Grzymala-Busse, J.: Data with missing attribute values: Generalization of indiscernibility relation and rule induction. *Transactions on Rough Sets* 1, 78–95 (2004)
5. Kowalski, M., Stawicki, S.: SQL-based heuristics for selected KDD tasks over large data sets. *Proc. FedCSIS 2012*, 303–310 (2012)
6. Orłowska, E., Pawlak, Z.: Representation of nondeterministic information. *Theoretical Computer Science* 29(1-2), 27–39 (1984)
7. Pawlak, Z.: Systemy Informacyjne: Podstawy Teoretyczne (in Polish) WNT (1983)
8. phpMyAdmin Web Page `http://www.phpmyadmin.net/` (2016)
9. Sakai, H., Okuma, A.: Basic algorithms and tools for rough non-deterministic information analysis. *Transactions on Rough Sets* 1, 209–231 (2004)
10. Sakai, H., et al.: Rules and apriori algorithm in non-deterministic information systems. *Transactions on Rough Sets* 9, 328–350 (2008)
11. Sakai, H., Wu, M., Nakata, M.: Apriori-based rule generation in incomplete information databases and non-deterministic information systems. *Fundamenta Informaticae* 130(3), 343–376 (2014)
12. Sarawagi, S., Thomas, S., Agrawal, R.: Integrating association rule mining with relational database systems: alternatives and implications. *Data Mining and Knowledge Discovery* 4(2), 89–125 (2000)
13. Ślęzak, D., Sakai, H.: Automatic extraction of decision rules from non-deterministic data systems: Theoretical foundations and SQL-based implementation. *DTA2009* Springer CCIS Vol.64, 151–162 (2009)
14. Swieboda, W., Nguyen, S.: Rough set methods for large and spare data in EAV format. *Proc. IEEE RIVF 2012*, 1–6 (2012)
15. Wu, M., Nakata, M., Sakai, H.: An overview of the getRNIA system for non-deterministic data. *Procedia Computer Science* 22, 615–622 (2013) `http://getrnia.org`